Assessing the Effects of Data Compression in Simulations Using Physically Motivated Metrics

D. Laney, S. Langer, C. Weber, P. Lindstrom, A. Wegener

April 30, 2013

Super Computing
Denver, CO, United States
November 17, 2013 through November 22, 2013

Assessing the Effects of Data Compression in Simulations Using Physically Motivated Metrics

Daniel Laney
Lawrence Livermore Lab
dlaney@llnl.gov

Steven Langer
Lawrence Livermore Lab
langer@llnl.gov

Christopher Weber
Lawrence Livermore Lab
weber30@llnl.gov

Peter Lindstrom
Lawrence Livermore Lab
pl@llnl.gov

Al Wegener
Samplify
AWegener@samplify.com

ABSTRACT

This paper examines whether lossy compression can be used effectively in physics simulations as a possible strategy to combat the expected data-movement bottleneck on future high performance computing architectures. This paper shows that, in a number of cases, compression levels of 2–5X can be applied without causing a significant change in the physical quantities that are of most interest for the simulation.

Rather than applying classical error metrics from signal-processing, we utilize physics based metrics appropriate for each code to evaluate the impact of compression. We evaluate simulations run with three different codes: a Lagrangian shock-hydrodynamics code, an Eulerian higher-order hydrodynamics turbulence modeling code, and an Eulerian coupled laser-plasma interaction code. We apply compression to relevant quantities after each time-step to approximate the effects of tightly coupled compression and also study the compression rates to estimate disk-bandwidth reduction. We find that the error characteristics of compression algorithms must be carefully considered in the context of the underlying physics being modeled.

Keywords

compression, high performance computing

1. INTRODUCTION

The computing power of large systems is increasing faster than the memory and disk bandwidth of the systems. A number of reports have indicated that this will lead to serious difficulties in effectively exploiting proposed Exascale (10^{18} floating point operations per second) systems. In particular, a large number of applications will be limited by memory bandwidth rather than processor performance [1, 2]. Applications with large memory footprints may be unable to write a checkpoint to disk in a time short compared to the mean time between application interrupt (MTBAI). The memory per core will also drop, forcing some strong scaling relative to current systems. It will be difficult to increase either disk bandwidth, memory bandwidth, or memory capacity above projections due to power constraints.

This paper examines whether compression can be used effectively in physics simulations. Compression allows more information per second to flow across a channel with a given bandwidth. We are interested in two possible uses of compression. In "disk compression," arrays are compressed before they are written to disk. In "memory compression," arrays are compressed while stored in off-chip memory and decompressed when they are loaded into on-chip cache for computation.

Preliminary studies showed that lossless compression does not deliver enough reduction in size to make a significant difference. Many physics simulations start out with large uniform regions. Lossless compression ratios are high for the initial conditions. Once the simulation starts running, the low order bits of the variables rapidly take on a semi-random character. For example, data arrays from LLNL's pF3D code can only be compressed about 15% using gzip. Lossy compression methods are necessary to have a significant impact on the memory or disk bandwidth requirements of a simulation.

While checkpoints are written to disk a few times per day, memory compression occurs several times per time step. Memory compression has a larger impact on the accuracy of a simulation than disk compression because it occurs hundreds of times more frequently. Our tests apply compression once per time step, which makes them a reasonable surrogate for memory compression.

Disk compression has no impact on the simulation when used for data files, but lossy compression will change the answer when used for checkpoint restart files. File formats such as HDF5 already provide support for lossless compression, and their flexibility would make it straightforward to also add support for lossy compression.

The bandwidth per core to the parallel file system on Cielo, a Cray XE-6 at Los Alamos National Laboratory, is roughly 1 MB/s when 64k cores are writing. Lossy compression algorithms have a throughput of roughly 100 MB/s, so the time spent compressing data is small compared to the time spent writing it to disk.

The memory bandwidth per core on an Intel Sandy Bridge is roughly 2.5 GB/s. It takes longer to compress the data at 100 MB/s than it does to transfer the uncompressed data between the memory and the cache. Using memory compression will cause a simulation to run more slowly on current systems, but it enables larger simulations to be run. The performance penalty for using compression will drop as the ratio of memory bandwidth to computation rate drops on future systems. Hardware compression logic on the processor chip could achieve much higher compression speeds, but will require a large market before being economically viable.

Depending on the chosen compression ratio, lossy compression will change the values in a given zone by an amount large compared to arithmetic roundoff. That is not a showstopper – many decisions made in setting up a job also change the answer. The physicist running a code picks a zone size based on a tradeoff between more accurate answers and the computational cost of the simulation. A physicist running a Monte Carlo transport simulation makes a choice of the number of Monte Carlo particles to use based on a tradeoff between the statistical accuracy of the answer and the computational cost of the simulation. The physicist usually makes these choices based on the desired accuracy of various integral physical quantities, not by bounding the expected zone-by-zone differences between two proposed simulations.

We suggest using the same integral physical quantities to assess the impact of compression. The code developer will run test problems with and without compression and evaluate key physical quantities. The results can be used to select the best compression algorithm and the proper compression level. This paper shows that, in a number of cases, 2–5X compression ratios can be applied without causing a significant change in the physical quantities that are of most interest for the simulation.

This paper uses simulations run with three different codes to evaluate the impact of compression. LULESH is a shock hydrodynamics mini-app. Miranda is a hydrodynamics code used for large turbulence simulations. pF3D is used to simulate the interaction of a high intensity laser and a plasma (ionized gas).

Lossy compression is most frequently used for photographs and for videos where key-frames periodically reset the error to zero. The physics codes we consider run time-dependent simulations where the errors can potentially accumulate. If the compression errors at one time step are strongly correlated with errors at the next time step, the cumulative effect might grow more rapidly than if they are uncorrelated. A compression algorithm should be judged both on the level of compression it achieves and the characteristics of the differences it generates.

The paper is organized into three sections. Each section begins with a high-level discussion that should be accessible to a general HPC audience. The rest of the section provides details that may only be of interest to experts. The final section summarizes our results and indicates topics for future research.

2. PRIOR ART

Numerical compression has an exceedingly rich history dating back to the 1960s. However, the compression research community has understandably focused on compression of consumer speech, audio, images, and video, given the ubiquity of such media. Media compression techniques are inappropriate for a universal numerical encoder because the quality metrics for media compression are determined by limitations of human hearing and vision, not by the accuracy requirements of numerical computations. Compressive sensing (CS) [3, 4] is targeted towards real-time integer sensor systems but requires significant back-end complexity to unwind the analog front end's random sampling. Waveform coders [5, 6] encode integer samples but can lack flexibility. Lossless compression of scientific data [7, 8, 9, 10] comes closest to a universal numerical encoder but typically achieves less than a 2X compression ratio on floats and even lower ratios for doubles. Furthermore, most such algorithms do not support lossy compression. The APAX algorithm [11, 12] has previously been applied to climate data [13], computed tomography x-ray samples [14], and a variety of integer and floating-point datasets [15].

The simulations presented in this paper operate repeatedly on data that has been previously compressed, leading to the possibility of amplification of compression induced differences. The earlier papers present results where compression was used on output to reduce the disk space required to archive results, but did not have any feedback on the simulation.

3. COMPRESSION ALGORITHMS

In this paper we study two compression algorithms designed for floating-point data: Samplify's *APAX* (APplication AXceleration) encoder [11, 12] and the *fpzip* [7, 16] compressor developed at LLNL. We describe these two compressors in parallel to highlight their similarities and differences.

In most physics simulations, the low order bits of floating point numbers are effectively random. The presence of these random bits prevents lossless methods from achieving high compression. Some of the low order bits of floating point numbers can be removed without a significant impact on physics answers. This results in compression at the expense of lost information. Once the low order bits have been removed, there is much more correlation between adjacent numbers. A lossless compression scheme based on encoding the differences between adjacent numbers is then applied. Applying lossless compression after zeroing the bottom 32 bits of a double precision number usually produces total compression ratios around 3–4X.

3.1 The fpzip and APAX Compressors

Although transform coding (e.g. using wavelet or discrete cosine bases) is traditionally used for lossy compression, both algorithms studied here have been designed to also support lossless compression. Such bit-for-bit lossless compression can be difficult to achieve using transform coding due to the subtleties of floating-point arithmetic, including rounding modes and error, catastrophic cancellation, order-of-evaluation dependence, extended precision, etc. General purpose lossless compression, as implemented for instance in *gzip* and *bzip2*, avoids such issues, but tends to be ineffective for floating-point data.

For these reasons and for speed, *APAX* and *fpzip* both rely on predictive coding, in which each floating-point number is predicted based on a trend of recently encoded values. The prediction *residual* (difference between actual and predicted value) tends to be small and can generally be encoded using fewer bits than the original floating-point value.

When used for lossy compression, *APAX* and *fpzip* both begin by *quantizing* the value f to be encoded, in effect reducing its precision. Quantization in *APAX* involves converting a *block* of N consecutive values (usually $N = 256$) to a signed integer representation, which can be thought of as aligning the floating-point values in a block so that they are represented using a common largest exponent, which is encoded with each block. If differences in floating-point exponents are large, this uniform quantization step may result in some loss of precision for the smallest (in magnitude) values in the block. After exponent alignment, each signed significand is treated as a 32-bit integer $\hat{f}$. For (64-bit) double-precision data, this implies that the bottom 32 bits of the significand are discarded. This design decision in *APAX* was motivated by performance reasons – by modifying *APAX* to use 64-bit integer arithmetic, such truncation could be avoided, possibly at the expense of slower compression. *APAX* then quantizes the integer $\hat{f}$ uniformly to $\bar{f} = \text{round}\big(\frac{\hat{f}}{q}\big)$, where q is a quantization level either specified by the user or computed adaptively by *APAX* to meet a target coding rate.

fpzip delays the integer conversion and leaves the values in their floating-point representation, quantizing the significand instead. *fpzip* restricts q to be an integer power of two, which effectively leads to truncation of the significand by discarding (zeroing) some fixed number of least significant bits. (Setting $q = 1$ guarantees entirely lossless compression.) This non-uniform quantization allows the *relative error* associated with lossy compression to be bounded in *fpzip*, whereas within each *APAX* block the *absolute error* is bounded. Quantization is the only potential source of loss in both compressors.

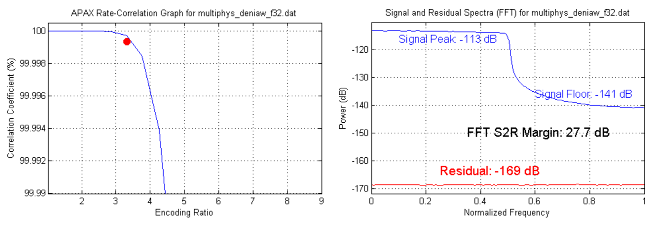
Following quantization, each value is predicted as a linear combination of recently encoded values. Both compressors rely on polynomial interpolation for prediction, with fixed integer polynomial coefficients. *APAX* uses univariate Lagrange polynomials of degree 0 and 1, allowing linear polynomials (or any function with $\frac{\partial^2 f}{\partial x^2} = 0$) to be reproduced. That is, the unknown value $\bar{f}[x]$ is predicted in terms of the n previous, known values by solving $\sum_{i=0}^{n}(-1)^i\binom{n}{i}\bar{f}[x-i] = 0$. The "best" polynomial degree $n-1$ is chosen locally by monitoring its effect on compression.

fpzip exploits correlations in more than one dimension using the Lorenzo predictor [17], which over a 3D domain reproduces trivariate quadratic polynomials (or any function for which $\frac{\partial^3 f}{\partial x \partial y \partial z} = 0$). In 3D this predictor solves $\sum_{i,j,k \in \{0,1\}}(-1)^{i+j+k} f[x-i, y-j, z-k] = 0$ for $f[x, y, z]$, and thus requires buffering a whole 2D "slice" from the 3D domain. Note that this predictor uses only additions and subtractions of known values.

Given the true floating-point value f and its prediction p, *fpzip* converts f and p to integers $\bar{f}$ and $\bar{p}$ via a monotonic mapping that treats the binary floating-point number as a sign-magnitude integer. This step is not needed in *APAX*, where the integer conversion occurs earlier. Both compressors then compute an integer residual $\bar{r} = \bar{f} - \bar{p} = s(2^e + d)$ with sign $s \in \{-1, 0, +1\}$, exponent e, and e-bit difference $d \in \{0, \ldots, 2^e - 1\}$. The bits of d generally exhibit no correlation, and are transmitted verbatim. However, the sequence of "signed exponents" $\tilde{e} = s(e+1)$ tends to be highly correlated (if not necessarily peaked around zero). *APAX* exploits spatial correlations by encoding differences between consecutive exponents in small groups. *fpzip*, on the other hand, models the non-uniform distribution of exponents and encodes each $\tilde{e}$ independently using a fast entropy coder.

3.2 APAX Profiler

A compression algorithm that takes advantage of "signal" characteristics may provide better compression ratios than one that does not. The spatial wavelength dependence of data arrays may be examined with popular applications like MatLab and Mathematica. APAX includes a profiler that allows a user to investigate the compressibility of their data and APAX's tradeoffs between bit rate and signal quality. Figure 1(a) illustrates the APAX rate-distortion curve of the pF3D deniaw variable. The profiler suggests a Recommended Operating Point (ROP) where the Pearson's correlation coefficient between the original data x and the APAX-decoded data $\tilde{x}$ is 0.99999 ("five nines"). In Figure 1(b), the profiler compares the input signal spectrum (upper curve) to the residual spectrum (lower curve) and quantifies the spectral margin at the ROP.



(a) (b)

Figure 1: The figure shows the APAX profiler results for the deniaw array from a pF3D simulation.

4. SIMULATIONS AND VALIDATION METHODS

4.1 LULESH

LULESH is a shock hydrodynamics mini-application developed for use in evaluation of current and future computer systems and proposed programming models [18, 19, 20]. LULESH solves the Sedov blast wave test problem – a point explosion surrounded by an initially uniform surrounding gas. The gas consists of one material and is modeled in three dimensions using a Lagrangian (moving mesh) formulation. Figure 2 depicts two key features of the simulation – the deformation of mesh elements and the shock wave.

LULESH solves the inviscid compressible Navier-Stokes equations in the Lagrangian formulation. A staggered mesh approximation [21] with single point quadrature for element-centered thermodynamic quantities such as density and pressure is utilized. Kinematic variables such as position and velocity are defined at mesh nodes. The Sedov problem presents an interesting use-case for compression, as most field values vary slowly over most of the domain, but change quickly and abruptly near the shock.

For the Sedov problem, the two key physics requirements are that the blast wave should be spherical, the shock radius versus time should match the analytic solution. In this paper we evaluate the symmetry of the shock by comparing the field values as a function of radius for various compression levels and methods, to the results obtained with a double precision simulation with no compression. We assess physical accuracy by estimating shock position as the distance to the centroid of the element with maximal density,

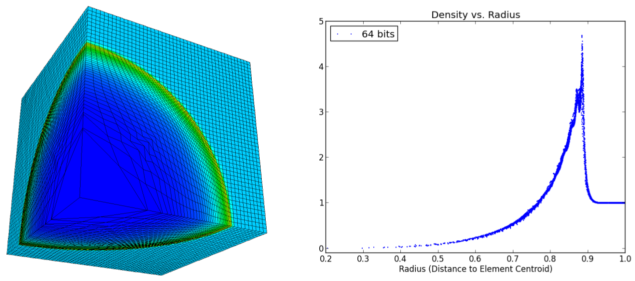


Figure 2: An overview of the LULESH shock-hydrodynamics simulation. (left) Density field showing the shock wave and mesh deformation. (right) Scatter plot of density vs. radius for every element in the default mesh size of 45 elements per side.

and comparing that value between runs with and without compression. Although this measure is not the most sophisticated way to assess shock position, it is simple can be consistently applied across LULESH runs at the same resolution. Finally, we measured differences in internal energy between compression and non-compression runs to provide validation that compression wasn't violating the expected behavior of the LULESH simulation as represented by full double precision runs at different mesh resolutions.

4.2 pF3D

The National Ignition Facility (hereafter NIF; [22, 23]) is a large NNSA experimental facility that houses the world's most powerful laser. One of the key goals of the NIF is to compress a target filled with deuterium and tritium to a temperature and density high enough that fusion ignition occurs.

The intensity of NIF beams exceeds $10^{15}\mathrm{W/cm}^2$ in the brightest spots. When intensities are this high, it is possible for the laser to couple to fluctuations in the plasma density and backscatter a significant fraction of the laser light. pF3D [24, 25, 26] is a multi-physics code which simulates interactions between laser beams and the plasma in NIF experiments. pF3D is used to evaluate proposed target designs and pick the ones with acceptably low levels of backscatter.

pF3D zones are roughly the size of the laser wavelength (0.35μm) while the plasmas of interest are several mm across. Simulations of the full path of a single laser beam require 50 billion or more zones. Simulations of five interacting beams may require more than a trillion zones. pF3D has been used to run simulations with 64k or more cores on IBM Blue Gene/L, IBM Blue Gene/P, and Cray XE-6 systems and work is underway to run a million core simulation on an IBM Blue Gene/Q system.

Figure 3 shows the intensity pattern of the laser beam as it enters the target in a NIF experiment. The bright spots are referred to as speckles. Speckles are narrow transverse to the laser propagation direction and extend for many laser wavelengths in the propagation direction. The laser is designed so that the speckles in the beam move around in time. The plasma temperature and density respond to the intensity averaged over time, so they see a smooth beam. The backscattered light is initially generated in the brightest speckles and grows in strength faster than the speckles move.

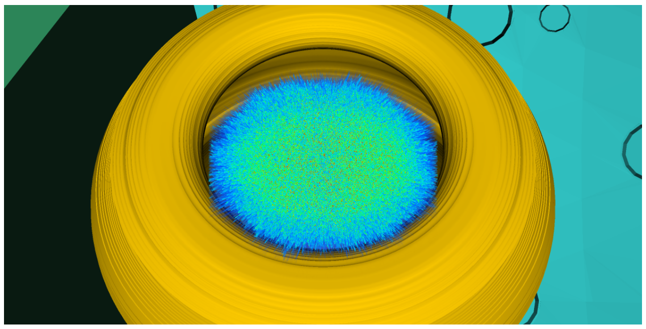


Figure 3: The figure shows the intensity of the laser beam as it enters the hohlraum. The hohlraum is a can-like object with the fusion target at its center. The intensity increases as the color changes from blue to green to yellow to red. The bright spots embedded in a lower intensity background are a design feature of NIF beams.

The laser light will refract (bend) when there is a density gradient. Speckles are long enough that a small change in density is enough to shift a speckle sideways by a significant fraction of its width. The fused multiply-add instruction of the PowerPC processor (it uses internal registers that are longer than 64 bits) produces results sufficiently different from x86_64 processors to cause speckles to move sideways by a zone or more. Compiler optimizations that change the order in which summations are carried out also cause speckles to move.

As a result of this high sensitivity to small variations, validation of new versions of pF3D relies on comparisons of the total transmitted and reflected light as a function of time (the total light is insensitive to the exact placement of speckles) and on intensity histograms. This paper uses the same physics based validation methods to assess the impact of lossy compression.

4.3 Miranda

Miranda [27, 28] is a Navier-Stokes code used to simulate a range of hydrodynamic problems with higher-order accuracy. It uses spectral methods or compact differencing to resolve turbulent structures with minimal dissipation. Dissipation is added at high wavenumbers through the use of artificial fluid properties [29], which act as a large-eddy simulation (LES) subgrid model. Miranda has run simulations using over 64k cores on Blue Gene/L systems.

The Miranda test problem simulates the growth of the Rayleigh-Taylor instability (RTI). Simulations of the RTI typically start with small-scale perturbations on the interface separating two fluids of different densities. The high density fluid is on top of the low density fluid – an unstable situation. The perturbation amplitudes grow, neighboring perturbations merge, and, eventually, turbulent mixing occurs. This inverse cascade of scales, from the initial short wavelength perturbation to large wavelengths at late times, requires high-order accuracy to ensure relevant features are not removed through dissipative numerics or influenced by the amplification of numerical noise.

The test problem is physically unstable to all perturbation wavelengths. Filtering is employed to damp the growth of

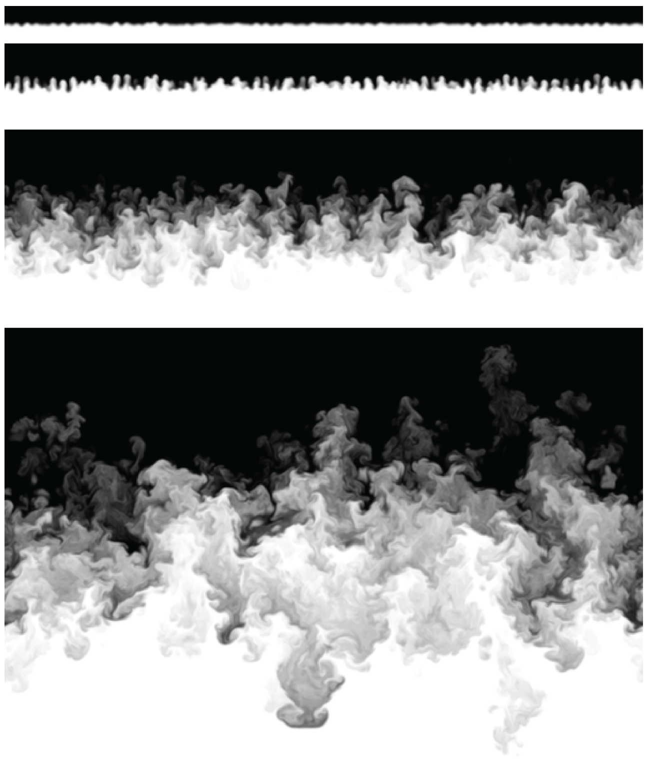


Figure 4: Density fields are shown from a Miranda simulation of the Rayleigh-Taylor instability at (from top to bottom) t/τ = 1, 2.5, 10, 20.

modes with short wavelengths. The damping is quite strong at a wavelength of 2 zones, but drops quickly as the wavelength increases. An important point to remember when investigating compression schemes is that the physics will amplify numerical perturbations introduced by compression as well as deliberately imposed perturbations. This means that the wavelength spectrum and step-to-step coherence of the perturbations produced by compression matter in addition to the compression ratio.

The incompressible simulations shown here are initialized with narrowband Gaussian perturbation spectra peaked at 8 grid cells per wavelength (λ_0) and an RMS amplitude of 0.1 grid cells. This allows the instability to begin in its linear stage. The two fluids have densities of $\rho_1 = 1$ and $\rho_2 = 3$, and gravitational acceleration of $g = 1$, providing a time scaling of $\tau = \sqrt{\lambda/Ag}$, where $A = (\rho_2 - \rho_1)/(\rho_2 + \rho_1) = 0.5$ is the Atwood number. Density fields from a $512^2 \times 1024$ simulation using 1024 processors are shown in Figure 4. In the first two images, at $t/\tau = 1$ and 2.5, the perturbations – just barely noticeable in the first image – are growing independently at an exponential rate. By $t/\tau = 10$, shown in the third image, perturbations have merged, producing larger scales and mixed fluid. At $t/\tau = 20$, shown in the last image of Figure 4, the layer has entered into an apparent turbulent state.

An important quantity in assessing RTI simulations is the mixing layer thickness, h, which is expected to behave as $\dot{h}^2 = 4\alpha Agh$ when the layer reaches a self-similar state at late times. Another important characteristic at late times is the spectrum of perturbations as a function of spatial frequency. These physically motivated quantities will be used to assess the differences between compressed and uncompressed simulations.

5. RESULTS

This section presents results on the acceptable level of compression in the LULESH, Miranda, and pF3D test problems. Simulations with and without compression are compared using the physics metrics mentioned in the earlier discussion. Simulations may be sensitive to the details of how compression is performed. As a result, the acceptable level of compression for a given test problem may differ between different compression algorithms.

The codes have been modified so that they compress and decompress variables at the end of each time step, with each variable represented on a (logical) Cartesian 3D grid. Compression would occur several times during each time step when using "memory compression" and only after several hundred time steps for "disk compression." We also ran a Miranda test where the compression function was called at the lower frequency of checkpoint dumps.

We report the minimum, maximum, and average compression ratios over all domains. The minimum compression ratio is the most important measure when using "memory compression." The process with the lowest compression ratio will have the hardest time fitting in the available memory and it will spend the most time reading and writing memory. All three of these codes use a bulk synchronous programming model, so the slowest process controls the performance. Efficient parallel I/O packages have performance that is dependent on the total number of bytes written, not on the number contributed by each process. With an I/O package of this sort, the average compression ratio is the important quantity.

5.1 LULESH

In this section we explore the effects of compression on the accuracy of the LULESH Sedov blast wave simulation. We invoked the compression algorithm at the end of each time step and recorded the compressed size. We then decompressed the data and stored it in the the simulation state for use in the next time step. We performed two studies: in the first study we varied the mesh size, and ran *fpzip* over a full range of precisions, to $t = 0.002$. In the second study, we ran with both $APAX$ and *fpzip* on a 105^3 mesh to time $t = 0.05$ at a small set of target compression rates ($APAX$) or precisions (*fpzip*). We assessed shock position in a manner similar to Tasker et al. [30] to compare the effect of the two compression schemes on the code. As we present below, we found that shock position did not vary greatly for most compression settings. In addition, the width of the shock was not a good indicator of the effects of compression, as this measure was not sensitive to noise in the data. In order to assess the more subtle effects of compression, we sorted all element centroids by distance from the origin to define fields as a function of radius, and and performed L_2 comparisons against simulations without compression.

In the first study, we ran a series of simulations with *fpzip* at precisions from 24 to 64 bits in steps of 4 bits on several mesh sizes. Figure 5(left) shows L_2 errors between the density versus radius curves for each compression rate and mesh size compared to the full precision result at the same mesh

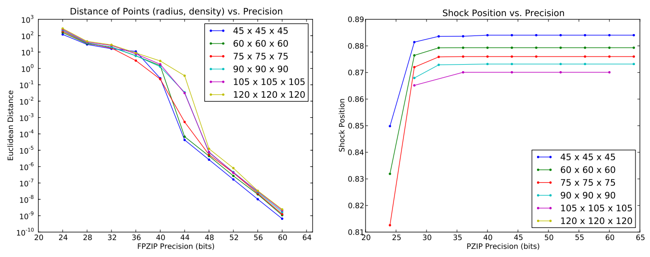


Figure 5: This figure summarizes the results of a parameter study over mesh size and *fpzip* precisions from 24 to 64 bits. (left) L_2 error of density field as a function of radius with respect to a run with no compression. For each run, density as a function of radius is assembled by sorting element centroids, and the L_2 error is computed. We conclude that precisions down to 48 bits maintain reasonable precision for this problem. (right) Shock position as a function of mesh size and *fpzip* precision. In this plot we see that as we increase mesh precision, the apparent change is shock position is larger than that induced by most precision settings. In this figure the shock position is not yet converged in LULESH.

compressor	ratio	energy	result
fpzip 64-bit	2.5 / 2.0 / 7.3	0.0	pass
fpzip 48-bit	2.6 / 2.0 / 14.0	$-1 \times 10^{-6}\%$	pass
fpzip 32-bit	4.9 / 3.2 / 20.8	-0.07%	pass
fpzip 24-bit	N/A	N/A	fail
APAX 2X	3.1 / 2.4 / 19.2	$+6 \times 10^{-6}\%$	pass
APAX 3X	3.5 / 3.0 / 19.2	-0.002%	pass
APAX 4X	4.5 / 4.0 / 19.0	-0.01%	see text

Table 1: This table shows the compression ratios achieved for the LULESH runs for mesh size 105^3. The 'ratio' column shows the total, minimum, and maximum compression ratios for all fields in the problem. The value for total compresion is the ratio between the storage required for all fields, divided by the total compressed sizes of all fields. The 'energy' column records the percent of internal energy lost (or gained), with respect to the full precision run. The compression algorithm and setting for each run is shown and is labeled by whether it passes the physics criteria.

sizes. There is a transition to larger errors at 40 and 44 bits of precision and this transition coincides with the field values becoming much more symmetric while at the same time the total solution becomes less spherical. This indicates that the loss of those bits is removing fundamental information from the simulation from which it can not recover, as we know that the Sedov problem has a spherically symmetric analytic solution. Figure 5(right) shows that mesh size has a larger effect on shock position than compression rate. Our assessment is that with respect to both shock position and L_2 error in individual density values, the LULESH Sedov simulation has correct behavior down to 48 bits, and that in several aspects precisions as low as 32 bits are sufficient. We confirmed this assessment by assessing internal energy loss as we present below.

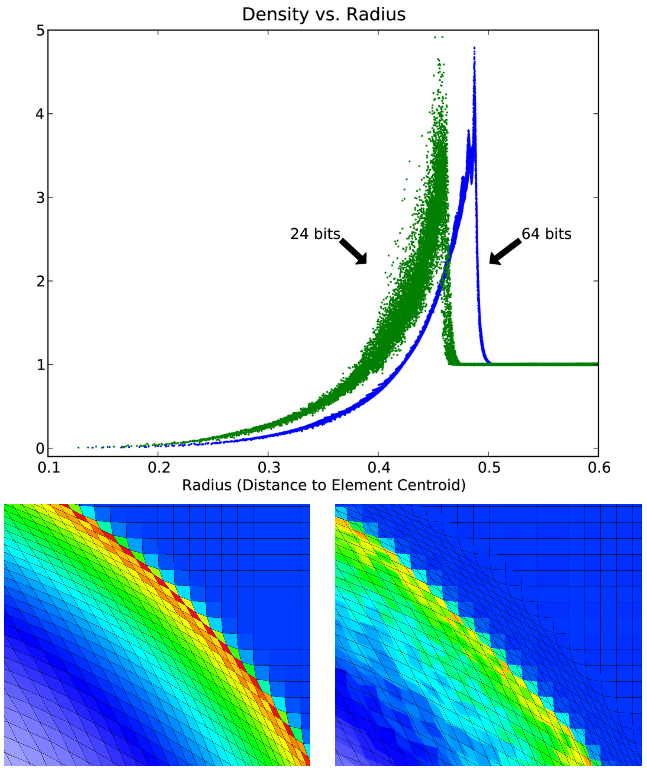


Figure 6: (top) Density plotted against radius at $t = 0.02$, for *fpzip* precision values of 24 and 64 bits, for a 105^3 mesh size. When retaining only 24 bits of precision, a significant amount of energy is lost, causing the shock to be delayed. (bottom left) A zoom in of the density field in the full precision case. (bottom right) The shock at 24 bits of precision, showing that a significant amount of noise is introduced, although this noise is symmetric. In both images, blue indicates low density, and red indicates the highest density.

Figure 6 provides an overview of the second study, showing density versus radius for *fpzip* precisions of 24 and 64 bits (lossless), as well as zoomed in views of the density field at these two precisions. In the second study we ran the simulations to t $t = 0.05$ so that the shock had traversed to the boundary of the mesh. This allowed us to gauge compression performance, since at early times the field values over the mesh are nearly constant before the shock reaches them. This study showed that the shock position was accurately captured at *fpzip* precisions down to 32 bits, and *APAX* rates up to 4X. We illustrate a failure mode in Figure 6(lower right) by showing the scatter in the shock position with 24-bit *fpzip* precision, indicating that it is no longer spherical. The shock radius is smaller than in the higher precision runs, indicating that energy has been lost and the shock is trailing the converged full-precision solution. By default, *fpzip* may introduce a bias as it first discards significand bits, effectivly rounding towards zero that may cause a loss of energy and mass. However, we also found that at low compression rates *APAX* also loses energy, presumably due to the scaling process it uses to convert floating point values into 32-bit integers before compressing them.

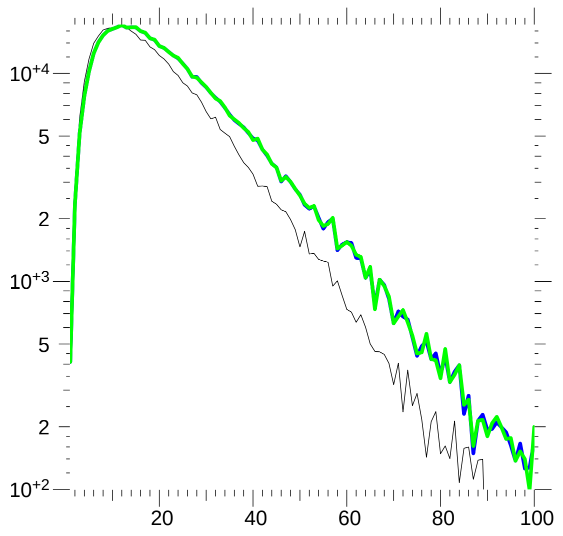


Figure 7: The figure shows a histogram of the amount of laser energy per bin as a function of the laser intensity. The beam has crossed the full extent of the plasma. The black curve is from a run using 24-bit *fpzip*, the blue curve is from a run using 32-bit *fpzip*, and the green curve is from a a run using uncompressed double precision variables. The blue and green curves overlay one another.

Table 1 shows the compression ratios and final internal energy change achieved for LULESH runs using both *fpzip* and *APAX* compression. The values are taken from $t = 0.05$. At the time that the data is computed, the field profiles are no longer constant everywhere, but instead vary across the volume which the shock has traversed. We see that over a fairly large range of compression rates / precisions that minimal energy is lost. We note that since *APAX* achieves a nominal 44-bit precision we might expect similar compression rates to *fpzip* at that precision, and we see that *APAX* compression rates fall between the *fpzip* rates at 32 and 48 bits. The 4X *APAX* run showed good agreement with internal energy and shock position but seeded non-physical noise in the nodal position fields ahead of the shock, so we did not deem that a total 'pass', nor a total 'fail'. The Sedov problem is stable, and it appears that this noise in nodal positions in front of the shock does not impact the shock position adversely, at least up to the time we ended the simulation.

It should be noted that the Sedov problem is a simple test case, and that the requirements of simulations involving more complicated geometries and multiple materials may have more restrictive requirements with respect to lossy compression. We also note that since LULESH is a mini-app it has stand-in data movement that does not impact the simulation, and we ignored those fields when reporting compressionr results. We also note that LULESH does not converge to a shock position cleanly, and up to mesh resolutions of 175^3 we saw the shock position continuing to vary slightly.

5.2 pF3D

Figure 7 shows a histogram of the amount of laser energy per bin as a function of the laser intensity. The histogram is made after the laser has crossed the full extent of the plasma, so energy has been lost due to absorption and to backscattered light. All of the pF3D simulations used double precision arithmetic. The blue (32-bit *fpzip*) and green (lossless *fpzip*) curves are effectively identical. The black curve (24-bit *fpzip*) is significantly lower at all but the lowest intensities. Most of the backscattered light is generated in high intensity regions, so there should be less backscatter for 24-bit *fpzip*.

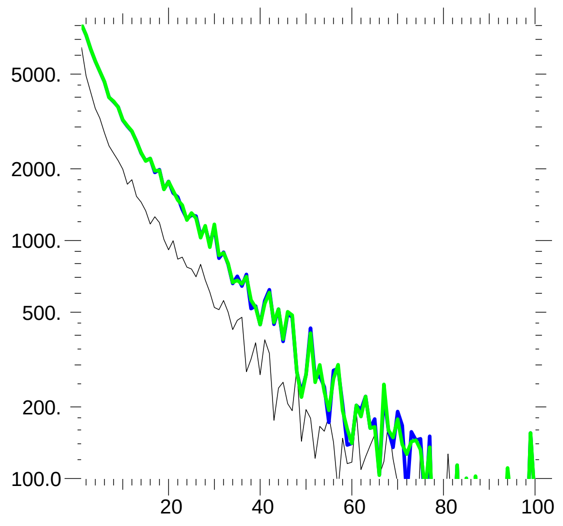


Figure 8: The figure shows shows a histogram of the amount of backscattered energy per bin as a function of the laser intensity. The histogram is made at the entrance plane where the backscattered light is at its maximum. The black curve is from a run using 24-bit *fpzip*, the blue curve is from a run using 32-bit *fpzip*, and the green curve is from a a run using lossless *fpzip*. The blue and green curves overlay one another. The black curve has lower energy at all intensities.

Figure 8 shows a histogram of the amount of backscattered energy per bin as a function of the intensity at the entrance plane. The blue (32-bit *fpzip*) and green (uncompressed) curves are effectively identical. The black curve (24-bit *fpzip*) has, as expected, significantly lower energy at all intensities.

Table 2 shows the compression ratios achieved for the pF3D runs. The 16- and 24-bit *fpzip* runs failed the physics criteria. The successful 32-bit *fpzip* run had a compression ratio of 3.66X. We also ran pF3D with float precision and no compression. This run also passed the physics criteria and could be thought of as achieving 2X compression. The run with 48-bit *fpzip* also achieved roughly 2X compression, but had higher precision than the run using float precision.

5.3 Miranda

To compare the effects of lossy compression on the development of the RTI, simulations with various levels of compression were run on $512^2 \times 1024$ grids using 1024 processors. After each time step, the density and the three components of velocity are compressed and decompressed using *fpzip* or *APAX*. This results in ~10,000 lossy compression steps over the course of the simulation.

In the case of *fpzip*, the compression step truncates the 64-bit data to either 48, 40, or 32 bits. Larger compression

compressor	ratio	energy	result
fpzip 64-bit	1.29 / 1.27 / 1.32	0	pass
fpzip 48-bit	1.91 / 1.86 / 1.97	0	pass
fpzip 32-bit	3.66 / 3.46 / 3.89	-0.2%	pass
fpzip 24-bit	6.57 / 5.89 / 7.37	-37%	fail

Table 2: This table shows the compression ratios achieved for the pF3D runs. The compression algorithm and setting for each run is shown. The 'ratio' column contains total, minimum, and maximum compression rates. The total state on each processor is summed and the rate on each processor computed. The min and max values represent the spread in information content across the domains of the simulation. Each run is characterized as passing the physics criteria, failing the criteria, or crashing. The 'energy' column shows the percent difference in the backscattered energy relative to the uncompressed run.

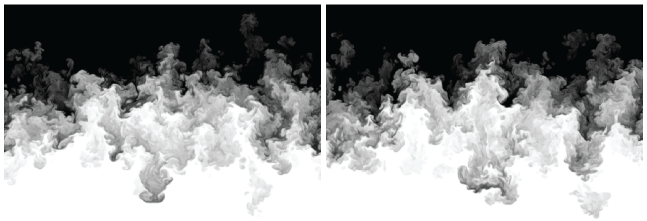


Figure 9: Density field at the end of the simulation from the reference (64-bit) simulation (left) and the *APAX* 5X case (right). Both have have nearly the same mixing-layer thickness and turbulence characteristics.

rates are achieved (~4X) in regions where the flow-field is relatively uniform (lossless compression is applied after the numbers are truncated). To test the effect of compressed checkpoint files, an additional test was run calling the 32-bit compression step every 500$^{\text{th}}$ time step, resulting in 16 compress/decompress calls over the course of the simulation. In contrast, *APAX* was used in three simulations with 2X, 4X, and 5X compression at each time step.

Images of the mixing layer at the end of the simulation ($t/\tau = 22$) are shown in Figure 9 from the 64-bit case (left) and the *APAX* 5X case (right). While there are subtle differences between the two images, the quantitative metrics presented below show that the integral quantities and the turbulence state are nearly identical.

The thickness of the mixing layer, h, is plotted in Figure 10 as the lower group of lines. The reference simulation (64-bit) is plotted in black while the 40- and 32-bit *fpzip* compression cases (called each time step) are plotted in blue and red, respectively. The *APAX* 5X case is plotted as a green dashed line. The other four cases considered are all plotted in gray, since their differences with each other and with the 64-bit curve are minor. The simulations were stopped once the dominant wavelength approached the size of the simulation domain. The 32-bit *fpzip* case is the only one that failed outright, as it crashed at $t/\tau \approx 14$ when it became numerically unstable. The other cases, and even the 32-bit *fpzip* case before crashing, differed little from the reference

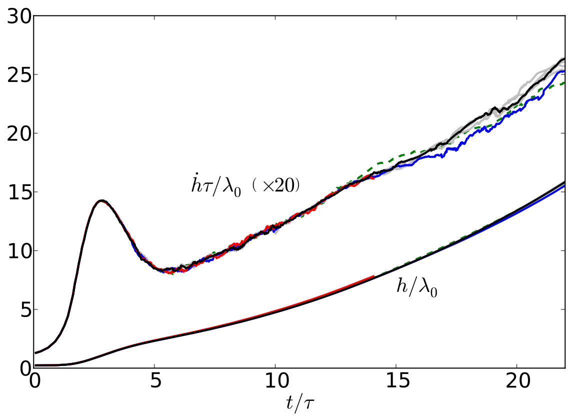


Figure 10: Mixing layer thickness (lower curves) and growth rate of the mixing layer (upper curves) from a reference (64-bit) calculation (black), an *fpzip* 32-bit compression case (red), an *fpzip* 40-bit compression case (blue), and an *APAX* 5X case (green dashed). The grey curves include a 48-bit *fpzip* case, a 32-bit *fpzip* case where compression is applied every 500$^{\text{th}}$ step, and *APAX* with 2X and 4X compression at every time step. Small differences are observable in the 32- and 40-bit *fpzip* cases. The 32-bit *fpzip* case crashed at $t/\tau = 14$.

simulation. At the time the 32-bit *fpzip* case crashed, its mixing-layer thickness was 2.4% larger than the reference simulation. The 32-bit *fpzip* case where compression was only applied every 500$^{\text{th}}$ time step ran without issue and differed by 0.38% in mixing-layer thickness at the end of the simulation ($t/\tau = 22$). The growth rates are also shown in Figure 10 as the upper set of lines with the same color scheme. At early times, when the layer is growing exponentially, there are no differences between the seven curves. Once nonlinear growth begins, after $t/\tau > 3$, small differences are noticeable. These differences are minor and all cases show the expected $\dot{h} \propto t$ self-similar growth beyond $t/\tau > 6$. Near the end of the simulation the *fpzip* 40-bit case and the *APAX* 5X case had 5-7% smaller growth rates, resulting in 1.9% and 0.49% smaller mixing-layer thickness, respectively. Since these growth rates are systematically smaller, rather than simply having different random fluctuations like the other cases (gray curves), it is likely that the mixing-layer thickness difference will compound and become unacceptable if the simulation were run later in time (requiring a larger initial domain). The differences in mixing-layer thickness at the end of the simulation are reported in Table 3.

The two-dimensional energy spectrum from the plane centered within the mixing layer is shown in Figure 11 at $t/\tau = 22$ for both the vertical velocity (solid) and the density (dashed) (the 32-bit case is not shown, as it did not reach this time). Aside from small fluctuations, the curves are nearly identical and all exhibit a -5/3 power-law for over a decade in wavenumbers, signifying turbulent behavior.

To further investigate why the *fpzip* 32-bit case crashed, Figure 12 shows density spectra at $t/\tau = 14$. The top set of curves show the energy spectra of the density field at the center of the mixing layer from all of the cases considered. At

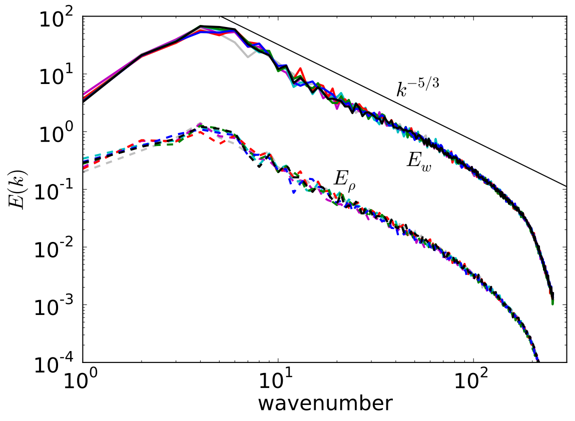


Figure 11: Energy spectrum of vertical velocity (solid) and density (dashed) at the mid-plane of the mixing layer at $t/\tau = 22$. All cases are shown except the 32-bit *fpzip* case, which did not reach this time. The spectra are very similar and feature a $k^{-5/3}$ inertial range.

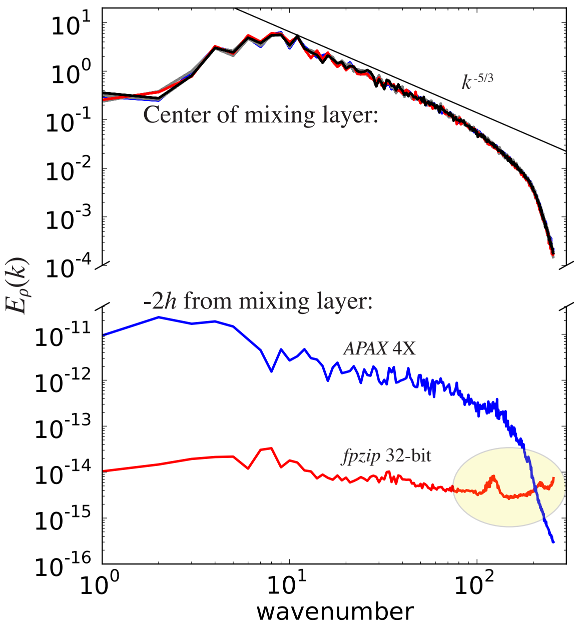


Figure 12: Energy spectrum of density at the mid-plane of the mixing layer (top set of curves) and at a distance $-2h$ from the mixing layer (bottom set of curves) at $t/\tau = 14$. The 32-bit *fpzip* case exhibits high wavenumber features that cause the problem to become numerically unstable.

this location the spectra are all nearly identical, showing the beginnings of a power-law range and a dissipation region at high wavenumbers caused by Miranda's LES filtering. The lower set of curves show the spectra from two simulations, taken from a plane located a distance $-2h$ from the center of the mixing layer, where the flowfield is relatively quiescent. The *APAX* 4X case produced a greater loss of precision than the *fpzip* 32-bit case, which is reflected in its larger

compressor		ratio	loss	thickness	result
fpzip	48-bit	3.2	5.9E-12	0.07%	pass
fpzip	40-bit	4.3	1.7E-09	1.9%	pass
fpzip	32-bit	4.7	4.1E-07	N/A	crash
APAX	2X	2.3	3.3E-10	0.007%	pass
APAX	4X	4.1	3.6E-06	0.19%	pass
APAX	5X	5.3	4.7E-05	0.49%	pass

Table 3: This table shows the compression ratios achieved for selected Miranda runs. The compression algorithm and setting for each run is shown and is labeled by whether it passes the physics criteria, fails the criteria, or crashed. The 'loss' column reports the RMS difference in the density field due to lossy compression divided by the mean density. The 'thickness' column shows the percent difference in mixing-layer thickness at the end of the simulation.

energy level in this region. In the *fpzip* 32-bit case, however, a bump can be seen in the high wavenumber region that corresponds to approximately 5 grid cells per wavelength. This scale is large enough to remain after Miranda's filtering routines. This spectrum also shows an up-turn at the highest wavenumber portion of the spectrum. It is these features that accumulate and interact with Miranda's numerics that cause the problem to crash. This shows that precision is not the only factor to consider when choosing a compression scheme for "memory compression," one must also consider, or test, how it will interact with the code's numerics.

The compression rate varied over the course of the simulation, within different regions of the simulation domain, and for the different fields being compressed. In the *APAX* 5X case, for example, the three velocity fields were all compressed to 5-5.5X, while the density field started out at 14X compression in regions far from the mixing layer, and then reduced to ∼5X compression later in time. The rates were averaged across time, domain, and fields and are reported in Table 3. Also reported in this table is the loss that compression introduced at each time step. This is defined as the RMS difference in the density field after compression was applied and normalized by the mean density. This loss criteria was evaluated in the center of the mixing layer. As noted in the table, all of the runs passed the physics criteria except for the 32-bit *fpzip* run. A short wavelength mode was seeded by compression in that run and the amplitude became large enough to crash the run.

The *fpzip* compression rate varied over the course of the simulation and at different regions within the simulation domain. Figure 13 shows the compression rate of all fields from a 1024 processor RTI simulation using Miranda and compressed with *fpzip* in 48-bit mode on each time step. The compression rate begins at ∼2.2 in processors near the mixing layer and at ∼4.5 in processors farther away. Over time the minimum compression rate decreases, reaching a floor in some processes at 1.7. Over the course of the simulation, an average compression rate of 3.3 was achieved.

Figure 14 shows the compression rate of one of the velocity components from a 1024 processor RTI simulation using Miranda and *APAX* 2X compression on each time-step. The variations in compression rate are much smaller than in the *fpzip* compression run. That is as expected. *fpzip* truncates numbers at a fixed bit position and then applies lossless com-

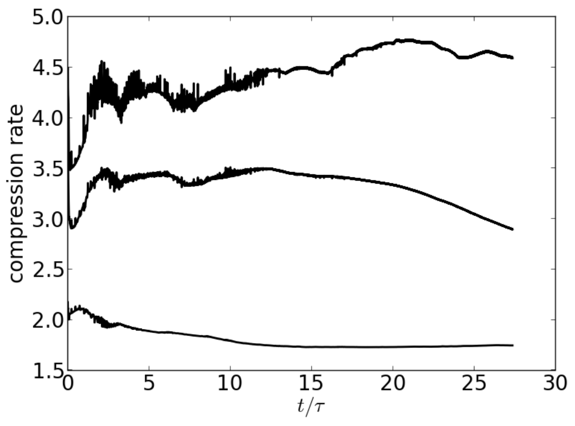


Figure 13: The compression ratio for an *fpzip* 48-bit compression run is shown. The three curves represent the maximum, minimum, and mean compression rate among the 1024 processes.

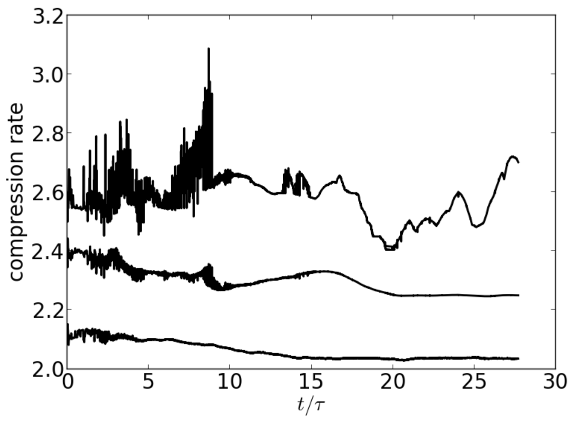


Figure 14: The compression ratio for a 2X *APAX* compression run is shown. The three curves represent the maximum, minimum, and mean compression rate among the 1024 processes.

pression. It achieves higher compression in regions where the variables vary slowly. *APAX* targets a specific compression rate. In regions where the variables vary slowly, *APAX* truncates fewer bits.

6. CONCLUSIONS

This paper has examined the impact of lossy compression on three physics simulation codes – LULESH, Miranda, and pF3D. All three codes can be run with a lossy compression ratio of 3X or greater when using a frequency suitable for "memory compression."

The LULESH study indicates that a broad range of compression parameters are valid for this run, and that there is an abrupt decline in L_2 errors of field values as function of radius around 40 and 44 bits precision. The Sedov blast wave problem is a stable problem, and we expect that for unstable flows high precision settings may be required. We also found that for LULESH, the mesh size changed the shock position more than the compression settings, until very low precisions broke down the spherical symmetry of the problem and seeded noise in the field values. Finally, we noted that *APAX* and *fpzip* were similar in their performance, indicating that compression schemes with either fix rate or fixed precision modes are viable. We found that with *APAX*, the per-block scaling of values did lead to unphysical noise in the nodal positions in front of the shock wave at a request compression rate of 4X. In future work, it would be interesting to thread the compression calls through the LULESH solver at the point when each field value is updated, as this would be a closer to how memory compression would occur in practice.

The Miranda run with 32-bit *fpzip* failed when compressed at every time step but passed when run at the much lower frequency required for "disk compression." This run had a compression rate of 4.7, only slightly higher than the successful 40-bit *fpzip* and *APAX* 4X runs. A pF3D run with 32-bit *fpzip* had low errors while a 24-bit *fpzip* run failed. These results show that the onset of failure occurs across a fairly narrow range of compression ratios.

Lossy compression is a viable approach to reducing the impact of limited disk bandwidth for all three codes. The much more frequent compression necessary if arrays are decompressed each time they are loaded into cache is also acceptable if some caution is used in the requested compression level. Software compression takes longer than it does to transfer the uncompressed data between memory and cache on Intel Sandy Bridge systems. "Memory compression" will hurt performance on current systems, but it will permit larger simulations to be run on a given number of nodes. Hardware compression schemes may be required to make "memory compression" a generally useful approach.

Acknowledgment

This document was released as LLNL-PROC-630714. This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-CONF-586774). Some of this material is based upon work supported by the Defense Advanced Research Projects Agency under its Agreement No. HR0011-07-9-0001.

7. REFERENCES

[1] S. Moore, "Multicore is bad news for supercomputers," *IEEE Spectrum*, vol. 45, no. 11, 2008.

[2] R. Murphy, "On the effects of memory latency and bandwidth on supercomputer application performance," *IEEE International Symposium on Workload Characterization*, pp. 34–43, 2007.

[3] R. Chartrand, "Nonconvex compressive sensing and reconstruction of gradient-sparse images: Random vs. tomographic Fourier sampling," in *IEEE International Conference on Image Processing*, October 2008, pp. 2642–2627.

[4] S. Gleichman and Y. Eldar, "Blind compressed sensing," *IEEE Transactions on Information Theory*, vol. 57, no. 10, pp. 6958–6975, October 2011.

[5] E. Ozturk, O. Kucar, and G. Atkin, "Waveform encoding of binary signals using a wavelet and its Hilbert transform," *IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 5, pp. 2641–2644, June 2000.

[6] D. A. Wright, "ADPCM coding and decoding techniques for personal communication systems," US Patent US 5 615 222, Mar., 1997.

[7] P. Lindstrom and M. Isenburg, "Fast and efficient compression of floating-point data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5, pp. 1245–1250, 2006.

[8] M. Burtscher and P. Ratanaworabhan, "High throughput compression of double-precision floating-point data," in *Data Compression Conference*, March 2007, pp. 293–302.

[9] N. Huebbe and J. Kunkel, "Reducing the HPC-datastorage footprint with MAFISC multidimensional adaptive filtering improved scientific data compression," *Computer Science Research and Development Journal*, vol. 28, no. 2–3, pp. 231–239, May 2012.

[10] E. Schendel, Y. Jin, N. Shah, J. Chen, C. S. Chang, S.-H. Ku, S. Ethier, S. Klasky, R. Latham, R. Ross, and N. Samatova, "ISOBAR preconditioner for effective and high-throughput lossless data compression," in *IEEE International Conference on Data Engineering*, 2012, pp. 138–149.

[11] A. Wegener, "Adaptive compression and decompression of bandlimited signals," US Patent US 7 009 533, Mar., 2006. [Online]. Available: http://www.patentlens.net/patentlens/patent/US_7009533/

[12] ——, "Block floating point compression of signal data," US Patent US 8 301 803, Oct., 2012. [Online]. Available: http://www.patentlens.net/patentlens/patent/US_8301803/

[13] N. Huebbe, A. Wegener, J. Kunkel, Y. Ling, and T. Ludwig, "Evaluating lossy compression on climate data," in *International Supercomputing Conference*, Jun. 2013, to appear.

[14] A. Wegener, N. Chandra, Y. Ling, R. Senzig, and R. Herfkens, "Effects of fixed-rate CT projection data compression on perceived and measured CT image quality," in *SPIE Medical Imaging Proceedings*, vol. 7627, Feb. 2010.

[15] A. Wegener, "Universal numerical encoder and profiler reduces computing memory wall with software, fpga, and soc implementations," in *IEEE Data Compression Conference Snowbird, UT (USA)*, March 2013, p. 528.

[16] P. Lindstrom, "fpzip version 1.0.1," 2008. [Online]. Available: https://computation.llnl.gov/casc/fpzip/

[17] L. Ibarria, P. Lindstrom, J. Rossignac, and A. Szymczak, "Out-of-core compression and decompression of large n-dimensional scalar fields," *Computer Graphics Forum*, vol. 22, no. 3, pp. 343–348, 2003.

[18] J. Keasler and R. Hornung, "Hydrodynamics Challenge Problem, Lawrence Livermore National Laboratory," Lawrence Livermore National Laboratory, Tech. Rep. LLNL-TR-490254, 2010.

[19] I. Karlin, A. Bhatele, B. L. Chamberlain, J. Cohen, Z. Devito, M. Gokhale, R. Haque, R. Hornung, J. Keasler, D. Laney, E. Luke, S. Lloyd, J. McGraw, R. Neely, D. Richards, M. Schulz, C. H. Still, F. Wang, and D. Wong, "LULESH programming model and performance ports overview," Lawrence Livermore National Laboratory, Tech. Rep. LLNL-TR-608824, December 2012.

[20] I. Karlin, A. Bhatele, J. Keasler, B. L. Chamberlain, J. Cohen, Z. DeVito, R. Haque, D. Laney, E. Luke, F. Wang, D. Richards, M. Schulz, and C. Still, "Exploring traditional and emerging parallel programming models using a proxy application," in *IEEE International Parallel & Distributed Processing Symposium*, May 2013, to appear.

[21] M. L. Wilkins, *Methods in Computational Physics*. Academic Press, 1964.

[22] E. I. Moses, "Overview of the National Ignition Facility," *Fusion Science and Technology*, vol. 54, no. 2, pp. 361–366, 2008.

[23] E. I. Moses, R. N. Boyd, B. A. Remington, C. J. Keane, and R. Al-Ayat, "The National Ignition Facility: Ushering in a new age for high energy density science," *Physics of Plasmas*, vol. 16, no. 041006, pp. 1–13, 2009.

[24] C. H. Still, R. L. Berger, A. B. Langdon, D. E. Hinkel, L. J. Suter, and E. A. Williams, "Filamentation and forward Brillouin scatter of entire smoothed and aberrated laser beams," *Physics of Plasmas*, vol. 7, no. 5, pp. 2023–2032, 2000.

[25] R. L. Berger, B. F. Lasinski, A. B. Langdon, T. B. Kaiser, B. B. Afeyan, B. I. Cohen, C. H. Still, and E. A. Williams, "Influence of spatial and temporal laser beam smoothing on stimulated brillouin scattering in filamentary laser light," *Phys. Rev. Lett.*, vol. 75, no. 6, pp. 1078–1081, Aug 1995.

[26] S. Langer, B. Still, T. Bremer, D. Hinkel, B. Langdon, J. A. Levine, and E. A. Williams, "Cielo full-system simulations of multi-beam laser-plasma interaction in NIF experiments," in *Proceedings of the 53rd Cray User Group Meeting*, 2011.

[27] A. W. Cook, W. H. Cabot, M. L. Welcome, P. L. Williams, B. J. Miller, B. R. de Supinski, and R. K. Yates, "Tera-scalable algorithms for variable-density elliptic hydrodynamics with spectral accuracy," in *ACM/IEEE Conference on Supercomputing*, 2005, p. 60.

[28] W. H. Cabot and A. W. Cook, "Reynolds number effects on Rayleigh-Taylor instability with possible

implications for type-Ia supernovae," *Nature Physics*, vol. 2, pp. 562–568, 2006.

[29] A. W. Cook, W. Cabot, and P. L. Miller, "The mixing transition in Rayleigh–Taylor instability," *Journal of Fluid Mechanics*, vol. 511, pp. 333–362, 2004.

[30] E. Tasker, R. Brunino, N. Mitchell, D. Michielsen, S. Hopton, F. Pearce, G. Bryan, and T. Theuns, "A test suite for quantitative comparison of hydrodynamics codes in astrophysics," *Monthly Notices of the Royal Astronomical Society*, vol. 390, no. 3, pp. 1267–1281, 2008.